Building a Scalable Infrastructure

Terence Critchlow

critchlow@llnl.gov
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

UC Davis
Aug. 17, 2000

UCRL-VG-140032

My background:

- 6 years experience in bioinformatics
- Ph.D. in Computer Science from Univ. of Utah
  - Worked on data transformation with the Utah Center for Human Genome Research
- LLNL
  - Working on the data integration with the Biology and Biotechnology program

Focus on applying computer science research to bioinformatics.

Goals for the class:

Outline the effects of design decisions on application scalability.

Identify some of the tools, techniques, and resources available.

Stimulate discussion about what constitutes a good bioinformatics design.

Outline

- **Define scalability**
- **General approaches**

} **Discussion**

- **Hardware scalability issues**
- **Software scalability issues**
- **Data exchange**

} **Discussion & Hands-on**

- **Wrap-up**

General format of class

- Lecture component
- General discussions
- Hands-on

Ask questions!

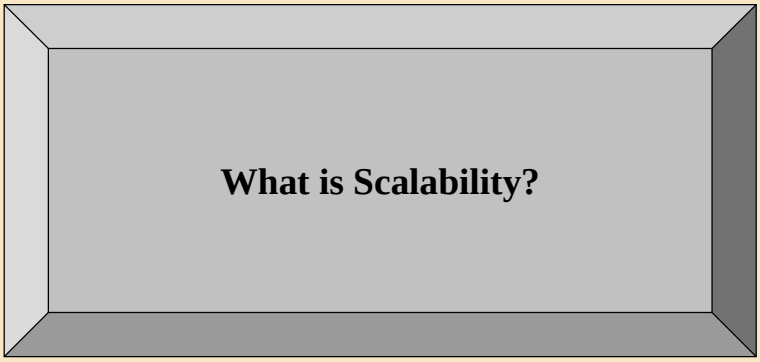
We will spend time on those areas of most interest to you.

A lack of questions will lead to a much less interesting class.

What is a bioinformatics system, architecture or infrastructure?

The collection of hardware and software used to support genomics research.

- PC / Mac
- Network
- Robotics
- Servers
- User interfaces
- LIMS
- Databases
- Analysis programs



What does it mean for a system to be scalable?

- **Merriam-Webster:**
  - **"Capable of being scaled"-up where scaled-up is**
  - **An increase according to a fixed ratio**
- **A better working definition:**
  - **A system capable of growing to meet future demands**
- **A practical definition:**
  - **A system in which fixing bottlenecks is easy**

Scalable systems are not perfect - just fixable.

Why do we need scalable systems?

- **We are unable to accurately forecast where the system will not meet demand.**
  - **Technology pushes applications in directions that were thought impossible just a year or two ago.**
- **Biology changes too fast for us to keep up using traditional approaches.**

> **"Biology evolves faster than computer science or technology, which is a scary truth indeed."**
> **Tom Slezak**


Contributing factors:

- **Users always want increased functionality. Now.**
  - **Deadlines can limit solutions.**
  - **As unplanned functionality is added, system complexity grows.**
- **Terminology is not consistent.**
  - **CS people and biologists/geneticists have trouble communicating needs.**
  - **Biologists from different fields use the same words in different ways.**
  - **New technology spawns new terminology.**


Why is this an issue?

- Prototype systems are often thrust into use without being “hardened”.
- Simplifying assumptions in the prototype limit the ability to handle real work-loads.
- Non-scalable systems become bottlenecks for the entire architecture.

Discussion:

What assumptions would you make designing an application for a single user to run on their desktop?

Some common assumptions:

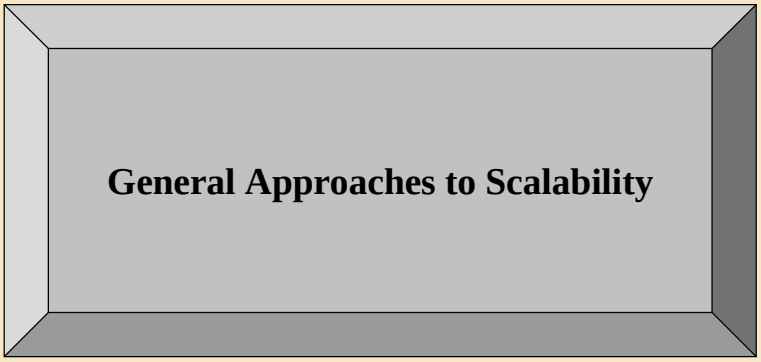
- Hardware probably a Mac or PC.
- Only one instance of the application running.
- Application and interface on the same computer.
- Certain software is available.
- Consistent use of terminology.
- Data is stored locally.
- Full control over available resources.
  - Memory, disk space, ….


How do those assumptions limit scalability:

- What would need to change if:
  - Your entire lab wanted to use it?
  - The entire company/university wanted to use it?
  - It was to be distributed to collaborators outside of your institution?
  - It was released over the Internet?

This is not an atypical progression for bioinformatics software.


General Approaches to Scalability



Components of a bioinformatics infrastructure:

Hardware

- **Network**
- **Specialty components**
  - **Robotics**
  - **Custom chips**
- **Computer**
  - **CPU**
  - **Memory**
  - **Disk**
  - **Operating system**

Software

- **User**
- **Applications**
  - **Analysis programs**
  - **Interface programs**
- **Data**
  - **Entry**
  - **Storage format**
  - **Access**

How do you design a scalable system?

Build a system that allows existing hardware and software components to be replaced, or new ones added, without affecting other components.

- Upgrade Memory.
- Add new disks.
- Switch hardware platforms.
- Upgrade software.
- Add new applications / interfaces.
- Switch database vendors.


What does this mean in practice?

- Modular design.
  - Limits the scope of an assumption
  - Allows components to be combined as needed
- Use of general purpose programming languages.
  - Provides hardware transparency
  - Do not use system specific features
- Use the right level of abstraction
  - Too detailed will make software too complex
  - Too abstract will not be able to solve the problem


What does this mean in practice? (cont)

- Question assumptions as you make decisions.
  - What will have to change if you are to move beyond the limits of that assumption.
- Paramaterize programs
  - Pass in constants instead of hard-coding them
- Make extensive use of meta-data
  - Data is easier to change than code
  - Generic code can be re-used in several places


Why take a modular approach?

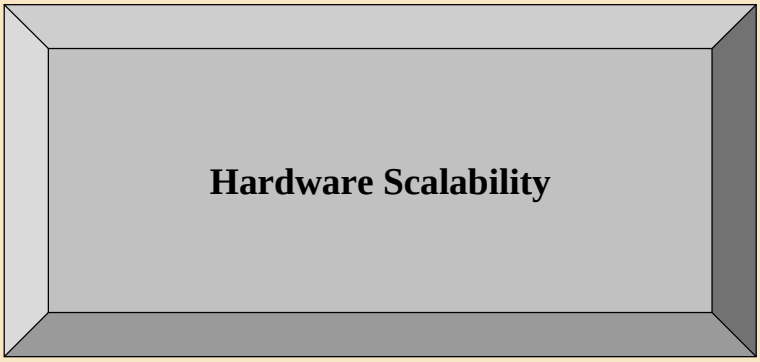
To meet tight deadlines, you will need to combine existing components in ways you never expected, while working around constraints you were told could not happen but somehow have.

Flexibility is the key to achieving scalability.


How do you remove bottlenecks?

- **Identify problem component.**
  - **Profiling**
- **Correct problem**
  - **upgrade / rewrite component**
  - **redesign component architecture**
  - **redesign system architecture**
- **Repeat**

Correcting one bottleneck only allows you to identify the next.



There are two major hardware components in a bioinformatics infrastructure.

- **Network**
  - Apple talk / Novell / Ethernet / ….
  - Bus / token ring / star / ….
- **Computer / Operating systems**
  - Mac / PC / workstation / server
  - MacOS / Windows / NT / UNIX / linux
- **Special purpose hardware**
  - Blast accelerators (Time logic, Compugen)
  - Super-computers
  - Robotics

Discussion:

What computer and operating system choices would you make?

Network recommendations:

- Ethernet (tcp/ip)
  - Allows connection to rest of the world
  - Common standard, so advances in technology will benefit you
  - Star or bus configuration (ring too slow)
- No restrictions on architecture
- Multiple vendors
- Easy to add machines (sub-nets)
- Fast and getting faster (1Gb Ethernet coming)


Server recommendations:

- Sun/HP/SGI server running UNIX
  - Multi-processor machine (if possible)
  - As much cache as possible
  - 1 GB main memory min
  - 100GB disk space min

  } Emphasize these over processor speed.
- Double minimum requirements if server both web and application server
- All components easily extended
- Multiprocessor allows large granularity parallelism
- UNIX designed for multithreaded, multi-user env.


Desktop recommendations:

- IBM compatible running NT
  - 128 MB ram min
  - 9 GB disk min
- More software available
- More robust than typical windows
- Easier to upgrade components than Mac

The most important thing is to have a consistent environment *if* at all possible.

Hands-On Exercise:

A Brief Intro to Unix

UNIX uses a combined window / command-line interface.

- Common commands
  - cd /home/critchlow - change directory
  - ls - list the contents of the directory
  - pwd - what is the present working directory
  - rm file.dat - remove (delete) file.dat
  - mkdir / rmdir foo - make / remove directory (folder) foo
  - man ls - list the manual (help) page for command ls
  - more file.dat - list file.dat to the current window

  - ps - shows all your running processes
  - kill 123 - stops process 123


Programs are executed from the command line prompt.

- Common tools
  - compress file.dat - compacts file.dat to conserve space
  - uncompress file.dat.Z - uncompresses file.dat.Z
  - tar -Bcf file.tar dir - creates file.tar (contains files in dir)
  - tar -Bxf file.tar - extracts all files from file.tar

  - perl - the perl interpreter
  - gcc - the gnu C/C++ compiler
  - emacs - file editor
  - ftp - file transfer program

http://wks.uts.ohio-state.edu/unix_course/intro-1.html


UNIX allows multiple programs to run concurrently (multitasking).

- By default commands run in the “foreground”
  - At most one program per window
  - Multiple windows may be open at the same time
- Placing “&” after a command runs it in the background
  - Allows other commands to be executed from the same window while it runs.
  - Processing continues even if window is closed.
- C-c stops a program
- C-z suspends a program
  - fg - bring process into foreground (resume)


UNIX uses common file extensions to imply file type (like windows).

- .Z - compressed (zipped) file
- .gz - gnu compressed (zipped) file
- .tar - tarred file
- .pl - perl file
- .c - C file
- .C or .cc - C++ file

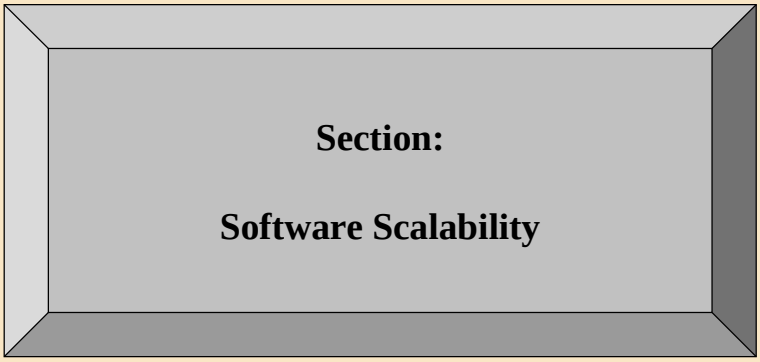

Emacs is a powerful text editor.

- **Basic commands:**
  - C-x C-f - open a file
  - C-x C-s - save a file
  - C-x C-c - close editor
  - C-s - search for word
  - C-d - delete character at cursor
  - C-v - move forward one screen
  - M-v - move backwards one screen
- **Available for multiple platforms**
  - Almost all UNIX flavors, windows

www.gnu.org

Discussion:

Questions about hardware issues?



There are three major considerations in a bioinformatics software infrastructure?

- Data management
  - Human / program / external data sources
  - Flat-files / Excel / Word
  - Relational / object-relational / object-oriented database
- Applications
  - Data processing
  - User interfaces
- User

Discussion:

Where does your data come from and where does it go?

Use a relational database to drive your applications!

- **Sybase**
- **Oracle**
- **Informix**
- **DB2**

Note: Excel, Word, MySQL, and flat-files are NOT on the list.

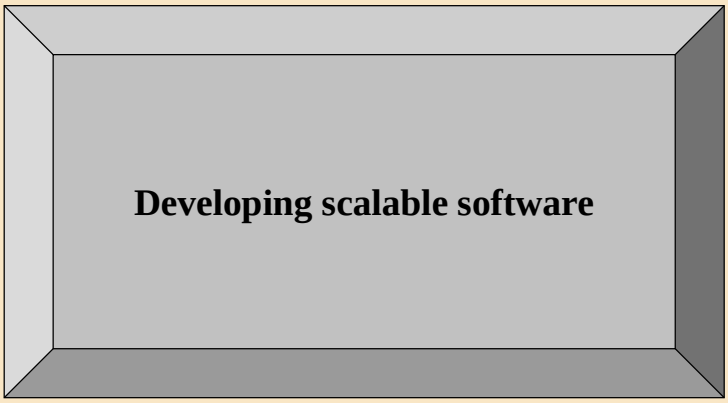
- **Databases provide:**
  - **Consistency**
  - **Query capabilities**
  - **Flexibility**
  - **Reliability**
  - **Parallel processing**
  - **Scalability**

Do not confuse the database with the user interface.

Using a database effectively requires experience.

- **Database configuration**
  - The number / size / location of partitions determines how much data you can have and how effectively it can be accessed
- **Schema**
  - Data layout determines how complicated queries are
- **Indices**
  - Proper indices ensure fast data retrieval.

http://www.compapp.dcu.ie/databases/welcome.html



Scalable application development requires programming for flexibility.

Every decision made during development affects scalability.

- Programming language
- Data flow
- User interface
- Application architecture
- Component granularity
- Buy vs. freeware vs. build

These decisions are closely related.


To select a programming language consider:

- How fast the application needs to run.
  - Compiled languages (C/C++) are generally faster than interpreted (Perl / Java)
- How long development can take.
  - Perl promotes rapid prototyping, C++ does not
- How portable the code needs to be.
  - Visual basic is only available on Windows
  - Most common languages (C/C++/Java/Perl) exist on all platforms
- How often the code will change.
  - Is speed more important than flexibility?


To determine data flow consider:

- Where each application gets its data.
  - Automatically generated / human entry
- Where the output needs to go.
  - Database / user / display / another program
- The desired input and output formats.
  - Output of one program needs to match input of next
- The opportunity to parallelize data flow.
  - Large scale parallelism is easy and often very effective


To choose an application interface consider:

- Command line interfaces.
  - Not very intuitive
  - Powerful
  - Can be batched
- GUI's.
  - Intuitive to users
  - Slow (particularly for experienced users)
  - Often machine specific
- web based interfaces.
  - Good way to reach distributed user group
  - Platform independent (theoretically, but not always)
  - Even slower


To select an application architecture consider:

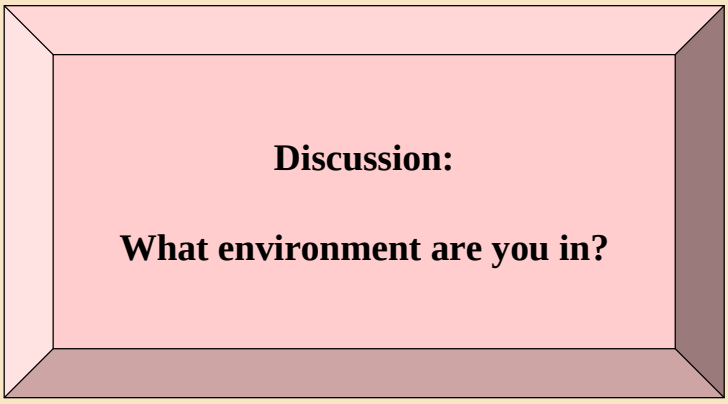
- The type of client and server machines available.
  - A large compute engine would favor a thin client for a small user base, but could become overburdened
  - If everyone has a Pentium-700, then a thick client or a stand-alone application on the desktop might be best
- The amount of data that needs to be moved.
  - Data intensive applications should be run on the same machine the data is stored on
- The location of your users.
  - Distributed user-base implies a web-based architecture


To determine component granularity consider:

- The smallest self-contained, useful functionality.
  - Should probably be represented as individual modules
- The cost of moving data between components.
  - If this is high combining modules may be warranted
- What functionality is likely to be replaced together.
  - Outdated code and bottlenecks will need to be replaced over time, so should form their own modules
- Potential gains from parallelizing the code.
  - Tasks need to be of a certain size before it makes sense to parallelize code


To decide on buy vs. freeware vs. build consider:

- If buying locks you in to a specific product.
  - If your architecture is flexible enough it shouldn't.
  - Is the cost (including support) reasonable ?
- If the free product is worth it.
  - Can you perform your own maintenance?
  - Does it fit into the rest of your architecture?
  - What assumptions are built in to it?
- If you have the resources to build it.
  - Can you meet the required deadlines?
  - Can you perform all of your own maintenance?


My recommendations:

- Oracle
  - OR technology may be useful in the long term
- Thin clients
  - Generally more scalable
- Java - for interfaces
- C++ - for time critical / heavy processing programs
- Perl - for everything else
- Use free stuff whenever you can
  - Avoid high cost programs, but don't re-invent the wheel
  - High-quality code is available for some applications


My recommendations:

- Web-based approaches are the current rage
  - Significant server hardware requirements
  - "Write once, run anywhere" programs don't
  - Impossible to address all needs this way

There is no right answer.

You need to look at the problem you are trying to solve within the context of your current environment and where you want to go.


Hands-On Exercise:

Using the Web

There are many search engines on the web:

- **Yahoo**
  - **http://www.yahoo.com/**
- **Alta-Vista**
  - **http://www.altavista.com/**
- **Excite**
  - **http://www.excite.com/**

Effectively using search engines is critical to getting the most from the web.

There is a lot of documentation available:

Use your favorite search engine to

- **Find information about Perl 5.**
  - **What does Perl stand for?**
  - **What is the comment character in perl?**
  - **http://www.perlreference.com/**
  - **http://www-tecc.stanford.edu/cgi-bin/perl-man**
- **Find the GNU Emacs users manual.**
  - **How do you create a new file in emacs?**
  - **http://www.gnu.org/manual/emacs-20.3/html_mono/emacs.html**


There are a lot of useful tools available:

- **RasMol**
  - **Protein structure viewer**
- **Blast**
  - **Local client and local database creation tools**
  - **Output parsers**
- **Special-purpose modules**
  - **bioperl - collection of Perl modules**
  - **biojava - collection of Java modules**
  - **biopython - collection of Python modules**

http://biocenter.helsinki.fi/bi/rnd/biocomp/


Many institutions distribute their data over the web:

- Find the home pages for
  - dbEST
  - PDB
  - SWISS-PROT
- Find the NCBI blast home page
  - http://www.ncbi.nlm.nih.gov/BLAST/

Discussion:

What sources do you use?

Answering some questions is still not easy.

- Given the sequence contained in file seq-1.dat find
  - The function of the resulting protein
  - The protein's secondary structure
  - The protein's structural classification
  - If the resulting protien was engineered or came from an organism(s)

Consistency is a problem

- Notice what happens when blasting that sequence against Swiss-Prot or PDB
  - Against PDB, you get 1DUZ which was engineered
  - Against SP, you get P01892 which points to several, including 1HLA which is from Human (notice that 1DUZ is not even listed)

While similar, there are differences between these chains. Can you tell, where the sequence came from originally?

How do web-interfaces work?

- Start page is usually written in html.
  - May be augmented with Java or Java-script
- Submitting query transfers data to the server.
  - There may be hidden data on a page
- Sever processes data.
  - CGI scripts are the most common way of doing this
  - Servlets are becoming more common
  - May connect to database, other programs, etc.
- Program returns new html page.
  - Usually, these pages are dynamically generated from a template based on query results


What is CGI?

- The common gateway interface.
  - Protocol for transferring data over the internet
  - Used primarily in web pages and http connections
- Data is passed as a string.
  - Parameter name-value pairs are passed
  - Limit on size of total string
  - Most languages have a parser module / class to convert to an internal representation
- Perl is the most common programming language for CGI scripts.
  - Usually small enough that speed is not critical


Discussion:

Are web interfaces scalable?

Aspects to consider:

- **Web server is doing most of the query processing.**
- **Asking a new question requires defining a new interface and support scripts.**
- **Interface designed for human interaction.**
  - **With some difficulty, programs can be made to call CGI scripts directly.**

Hands-On Exercise:

Beginning Perl

Lets look at a simple Perl program.

- Copy blast-1.pl to your directory
- Bring up Emacs
- Open the file
  - This program calls blast passing in a single sequence
  - The results are just printed to the screen
- Run the program from the command line

Discussion:

How would you make this program more useful?

Some suggestions:

- **Read sequence from file.**
- **Be able to handle multiple sequences.**
- **Select database on command line.**
- **Handle both protein and DNA databases.**
- **Parse the output into a different format.**
- **GUI interface ???**

XML has become the data interchange format of choice.

- Very simple format (similar to html).

  <tag> data </tag>

  <seq> AACCTGGGATTAAGG... </seq>

- Tags can be nested to form complex objects.

  <gene>

  <name>…</name>

  <seq>…</seq>

  </gene>

- An XML file has exactly one top-level object.

Why is XML useful?

- XML makes passing data easy.
  - Between programs
  - Between languages
  - Between computers
- Format of objects can be described in “DTD”s.
  - Allows identification of poorly formatted data
- XML parsers are freely available from the web.
  - Validating parsers compare data to a given schema
  - Non-validating parsers just look for correct syntax


Discussion:

What are some problems with XML?


Some suggestions:

- **Very verbose**
  - Hard to enter by hand
  - Takes up a lot more space
  - No binary representation
- **No semantics**
  - Tags no meaning, so it is easy to get confused
  - For example, I used the <seq> tag to refer to a DNA sequence. What if your program used it to refer to a protein sequence?
- **www.bioxml.org (GAME project)**
  - Developing a collection of dtds (schema) for bio data
  - As close to a standard as we have at this point


Hands-On Exercise:

Extending blast-1.pl


The results returned by blast-1.pl are not easily interpreted.

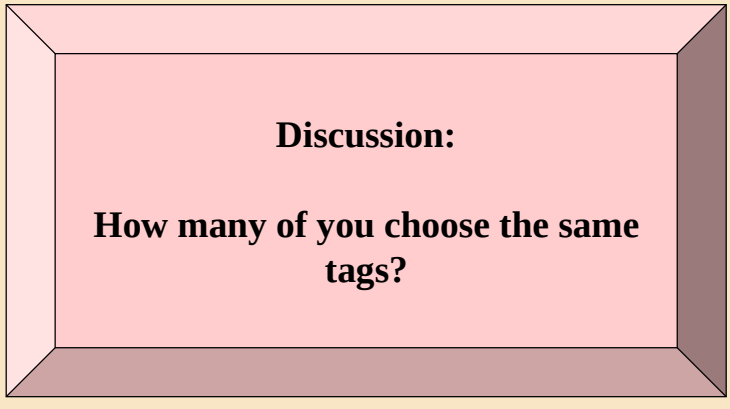
Modify blast-1.pl so that the blast results are printed in an XML format.

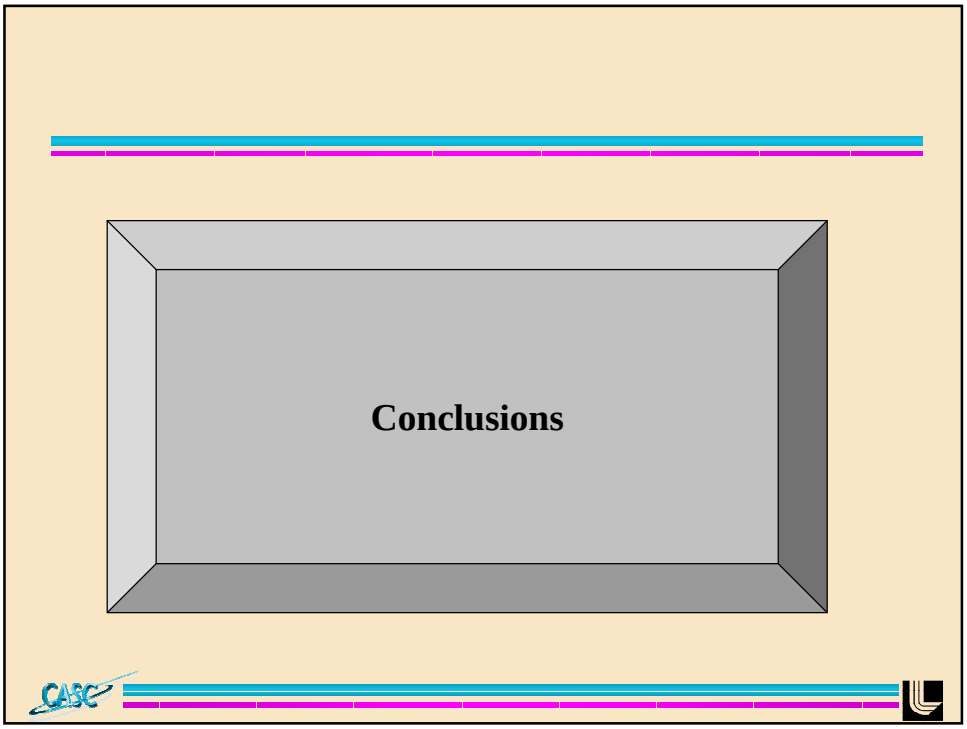
Novice Programmers

Start with blast-1b.pl

Experienced Programmers

Start with blast-1.pl





Other issues to consider:

- **Security**
  - Physical
  - Cryptography
  - Password
  - Firewalls
- **Social aspects**
  - Managers
  - Support
  - Funding
  - People
- **Legal issues**
- **Ethical issues**

What you should remember

- Building a scalable infrastructure is hard.
- Every decision you make has implications on scalability.
- Be as flexible as possible in your design.
- Use a relational database.
- Use existing software *where appropriate*.


What I tried to do:

- Give you a basic idea of how design decisions at all levels impact the scalability of the entire system.
- Expose you to the basic tools that you will need if you pursue bioinformatics further.

Convince you that building a scalable bioinformatics infrastructure is <u>not</u> easy.


For more information:

- **Web**
  - http://evol.nott.ac.uk/cmelun/links.html
  - http://www.hgmp.mrc.ac.uk/CCP11/
  - http://merlin.mbcr.bcm.tmc.edu:8001/bcd/Curric/welcome.html

UC Berkley Extension Classes

Bioinformatics Infrastructure Design
Programming for Bioinformatics

Tom Slezak

Homework due 5:00 Friday.

- **E-mail to critchlow@llnl.gov**
  - Just plain text, no word attachments please

A written report (100%) that provides:
* A definition of a scalable infrastructure
    * use at least two examples to highlight systems that are not scalable in different ways
* A detailed discussion of a bioinformatics problem where scalable infrastructures are needed. Make sure to address:
    * why this is a problem worth solving
    * what makes this a challenging problem
    * why scalability is a major hurdle to addressing this problem
    * the major components of this problem
    * limitations of current systems
    * How you would address this problem and why, including:
        * what tools you would use (and why)
        * how you would combine/integrate the problem's different components
        * why this approach is scalable

If you are familiar with an example that was not presented in class, please feel free to use it.

Discussion:

Questions / Comments?

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.